On-the-fly Interoperability through Automated Mediator Synthesis and Monitoring*

Antonia Bertolino[1], Paola Inverardi[2], Valérie Issarny[3], Antonino Sabetta[1], and Romina Spalazzese[2]

[1] CNR-ISTI, Pisa, Italy
[2] Università degli Studi dell'Aquila, L'Aquila, Italy
[3] INRIA, CRI Paris-Rocquencourt, France

Abstract. Interoperability is a key and challenging requirement in today's and future systems, which are often characterized by an extreme level of heterogeneity. To build an interoperability solution between the networked systems populating the environment, both their functional and non-functional requirements have to be met.
Because of the continuous evolution of such systems, mechanisms that are fixed a-priori are inadequate to achieve interoperability. In such challenging settings, on-the-fly approaches are best suited.
This paper presents, as an interoperability solution, an approach that integrates an automated technique for the synthesis of mediator protocols with a monitoring mechanism. The former aims to provide interoperability taking care of functional characteristics of the networked systems, whereas the latter makes it possible to assess the non-functional characteristics of the connected system.

1 Introduction

The realization of the Ubiquitous Computing vision [18] is still nowadays challenged by the often extreme level of heterogeneity in the system's underlying infrastructures, which in turns impacts on the ability to seamlessly interoperate.

Interoperability is a primary requirement in such systems, and, in order to achieve it, two aspects have to be considered: *functional interoperability* and *non-functional interoperability*. The first one solely refers to functional properties and aims at allowing the Networked Systems (NSs) to communicate. Instead, non-functional interoperability refers to the assessment and achievement of the non-functional characteristics which qualify the communication (*how* it should be provided). Indeed, while building an interoperability solution, both functional and non-functional properties of the connected system under-construction must be taken into account and ensured.

The fast pace at which technology evolves at all the abstraction layers additionally hampers the interoperability achievement between NSs in the digital environment. Interoperability should be "future-proof", i.e., NSs should be able to interoperate in spite of technological evolution and contextual changes.

* This work is partly supported by the CONNECT European Project No. 231167.

To face emerging functional and non-functional requirements in such heterogenous and evolving setting, relying on interoperability mechanisms that are fixed *a-priori*, can be proved to be inadequate, and *on-the-fly* approaches would be best suited.

Overcoming the interoperability barriers in the Ubiquitous Computing systems is at the heart of CONNECT [13]. The CONNECT Integrated Project aims at dropping the interoperability barrier by adopting a revolutionary approach to the seamless and eternal networking of systems, that is, by synthesizing on-the-fly the CONNECTors (or mediators) via which NSs communicate. The synthesis process is based on a formal foundation for CONNECTors, which allows learning, reasoning about, and adapting the interaction behavior of NSs at run-time. Synthesized connectors are concrete emergent system entities that are dependable, unobtrusive, and evolvable, while not compromising the quality of CONNECTed systems.

In this paper, as an excerpt of the CONNECT solution, we present an integrated approach to on-the-fly interoperability that combines automated mediator synthesis and monitoring. The former (based on a theory of mediator synthesis presented in [11]) aims at providing functional interoperability, whereas the latter takes care of non-functional interoperability.

One of the CONNECT underlying principle is to make minimal assumptions on the NSs. In particular, the project considers that NSs information, needed to compute an interoperability solution, is either declaratively provided by them or derived from them exploiting learning techniques. However, for the purpose of this paper, we assume NSs to come with their behavioral description and a set of policies describing their non-functional constraints, which may involve performance, dependability, security and trust requirements. In order to achieve a complete CONNECTion, the functional interoperability is pursued by construction through synthesis, whereas non-functional interoperability is addressed by suitably combining differing analysis, verification and enforcement techniques. An overview of the approaches under development in CONNECT for non-functional interoperability is given in [3]. In particular, we foresee to apply some approaches at synthesis time (see, e.g., the companion CONNECT paper by Di Giandomenico and coauthors [8]), in synergy with mediator construction. However, some operational constraints expressed as policies cannot be assessed statically at synthesis time. This paper focuses on this problem: we overview here the approach through which we combine the mediator synthesis with a runtime checking mechanism, implemented through monitoring.

Several other works in the literature relate to ours, both concerning the automated synthesis of mediators [19, 17], and monitoring [12, 5], especially in the context of service-oriented systems. The emphasis of this work however is *on the combination* of the two aspects and the approach we follow is mostly independent of specific technological frameworks.

The paper is organized as follows. We give an illustrative scenario and we present our approach at a high level (Section 2). Then, we recall our automated synthesis of mediators and we describe the integration with the monitoring of

mediators (Sections 3 and 4 respectively). Finally, we conclude with perspectives for future work (Section 5).

2 Approach description

This section outlines our approach by introducing a running example first, and then we explain the principles of our approach applied to that example.

2.1 Running example

Let us consider a *Photo-Sharing* system in a stadium; the system allows spectators to exchange pictures of the most significant happenings, e.g., goals in the case of football games. The spectators can be *producers* (respectively *consumers*) of pictures and hence they can *upload* (respectively *search, download*) pictures.

Different kind of interaction may be envisioned for the Photo-Sharing, including centralized and peer-to-peer ones. In a centralized implementation, the stadium would offer the Photo-Sharing service and the spectators' smartphones run service clients to upload, search, and download pictures; typical supporting middleware solution would be RPC-based, e.g., using a service-oriented middleware. In a peer-to-peer implementation, the spectators' smartphones would run a peer-to-peer application for photo exchanges (implementing the *upload*, *search* and *download* functionalities), for which a distributed shared memory *à la* tuple space would be the middleware of choice.

Fig. 1. High-level view of the Photo-Sharing NSs.

The behavior of the producers consists in *uploading* the photo, and the behavior of the consumer lies in *searching* and then *downloading* photos of interest.

Considering the shared memory implementation, the Photo-Sharing producer writes first the *metadata* and then the *file* associated with the given photo into the shared memory; while the consumer seeks the list of metadata descriptors matching a given *metadata* template into the shared memory and then iteratively reads the files of interest. With respect to the RPC implementation, the producer and consumer exchange photos calling the proper services on the server and receiving the corresponding replies. The producer calls the upload operation with both *metadata* and *file* as parameters. The consumer, instead, calls the search operation with a certain metadata and receives as reply the list of elements matching the request. Then iteratively the consumer calls the download

of selected pictures specified thanks to their identifier *ID* and receives as reply the corresponding files.

We consider, as a running example, the case in which an NS (NS_1) running a shared-memory-based photo producer protocol is in a stadium and the stadium is equipped with the shared-memory infrastructure. Another NS (NS_2) running an RPC-based photo consumer protocol accesses the stadium and wants to share pictures. Although apparently simple, this scenario presents substantial challenges to interoperability. In fact, despite the different implementations, the *intents* of the producers and consumers are *compliant* being upload and download pictures respectively. While there is an obvious behavioral (or protocol) mismatch between the RPC-based and the shared-memory implementations from the application down to the middleware layers. This type of mismatch can be addressed by emergent mediators, synthesized on the fly [11]. However, this not solve entirely the problem because constraints about non-functional properties are still not taken into account. Our proposal to manage these aspects is outlined in the following.

2.2 On-the-fly connector synthesis and monitoring

The networked systems[4] populating the open environment, e.g. the Photo-Sharing producers and consumers, are characterized by: *intent*, *behavioral description*, *ontological description*, and by *constraints* (expectations) about *non-functional* properties. The above characterization could be either declaratively advertised by the NSs or inferred exploiting learning techniques [10, 4].

The NSs constraints or requirements on the non-functional properties characterize "how" the interoperable connection should be provided. These constraints need to be expressed in a format that allows their automated interpretation and processing. In principle, different NSs could use many different languages to represent this information; in this paper we take the simplifying assumption that a mapping from such languages to a common CONNECT reference model exists and thus they are expressed in the same language.

To give an example of a NSs requirement, the photo-sharing consumer may require that the time it takes to get a list of photos that match a query be less than x ms.

We recall that a necessary condition for the networked systems to communicate is to be *compatible*. That is, to make sense for the NSs to communicate, they have to expose *complementary* (*provided*/*required*) intents. From a functional point of view, despite the complementarity of intents, the RPC-based consumer and the shared-memory-based producer cannot interoperate (communicate) directly with each other because their concrete protocols are different. In order to bridge this difference, a suitable *functional mediator* is synthesized on-the-fly [11]. The synthesis process happens at the time when the intention to communicate is manifested by either party. The goal of the synthesis process

[4] For the sake of simplicity, we explain the mediation process assuming only two NSs; in general, the same principles can be extended to scenarios with more NSs.

is to realize the "functional interoperability" by producing a protocol mediator that allows the two NSs to communicate dealing only with functional aspects.

In order to ensure that the functional mediator satisfies the non-functional constraints, we propose to couple the synthesized mediator with a suitable monitoring system whereby the non-functional constraints imposed by each NS can be checked at runtime. In this way, the monitoring is used to make sure that the connected system (i.e., the result of assembling the two NSs with the synthesized mediator) satisfies the expectations, in terms of non-functional characteristics, of both sides of the connection.

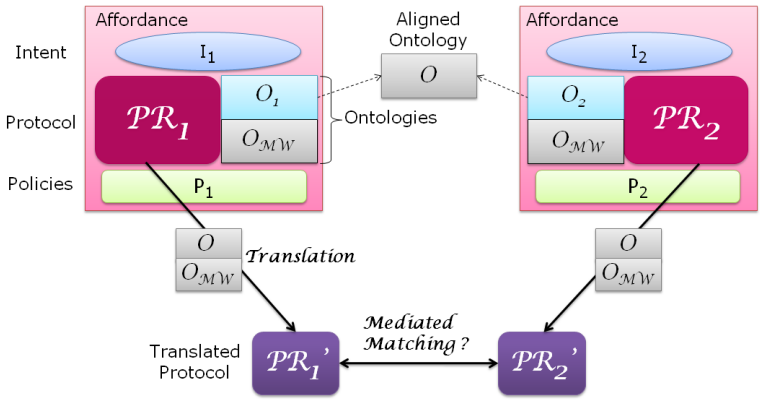

Fig. 2. An overview to mediators synthesis approach.

Thus, our approach addresses: 1) *functional interoperability* pursued by-construction at synthesis time (i.e.,*a-priori*), and 2) *non-functional interoperability*, that is compliance to non-functional constraints, continuously assessed at execution time (*a-posteriori*), by passive monitoring. Figure 2, whose elements are explained in the following, summarizes the main ingredients of our approach whose theory is presented in the companion paper [11].

We call *affordance* [9] the description of the functionality that is offered/requested by a networked system, i.e., to provide/require pictures in our Photo-Sharing scenario. In other words, an affordance is a high-level action-possibility (or functionality or capability) that characterizes the intended and/or possible interactions between the networked system and its environment.

We specialize this notion of affordance, to characterize our networked systems as mentioned in the beginning of this section.

More precisely, we consider that an affordance includes: (i) an *intent* (I_1, I_2 in Figure 2), that is used to perform a first check about the NSs compatibility in terms of complementarity of intents, i.e., provided/required functionalities; (ii) a *protocol* ($\mathcal{PR}_1$, $\mathcal{PR}_2$ in Figure 2), run by the system to carry on its capability, which is used to perform another check about protocol/behavioral compatibility;

(iii) *middleware* and *applications' ontologies* ($\mathcal{O}_{\mathcal{MW}}$ and $\mathcal{O}_1$, $\mathcal{O}_2$ respectively in Figure 2), describing protocol's actions exploited during the protocol translation done before the check of protocol compatibility; (iv) a set of *policies* (P_1, P_2 in Figure 2) that qualify the conditions that are required for the NS to function correctly. The policies are used to express constraints on the operational conditions under which a connection may take place.

In CONNECT certain types of policies, namely security policies, are not just checked but also enforced (see [7] for details). Other non-functional properties, such as those related to performance and reliability, business rules are assessed by passive observation on the live system and thus we can deal with them.

Our approach to the automated synthesis of mediators is briefly recalled in the next section, while monitoring integrated with the synthesis is the topic of Section 4.

3 Automated Synthesis of Mediators

In this section we introduce the necessary information about our synthesis approach to explain the integration with the monitoring. We summarize the automated synthesis of mediators that builds on the early theory of application-layer mediators presented in [16] and deals with the interoperability of both application- and middleware-layer. Additional details can be found in the companion paper on the theory of mediators [11] and in [1].

Given two NSs *affordances*, first we check (from a functional standpoint) that they have *compliant intents*, i.e. if they amount on the same capability (provided/required respectively).

Having checked the intent compliance, our goal is to synthesize a mediator to solve the *mismatches* occurring between the protocols. We use Labeled Transition Systems (LTSs) [14] to represent the protocols associated with the behavioral description of affordances.

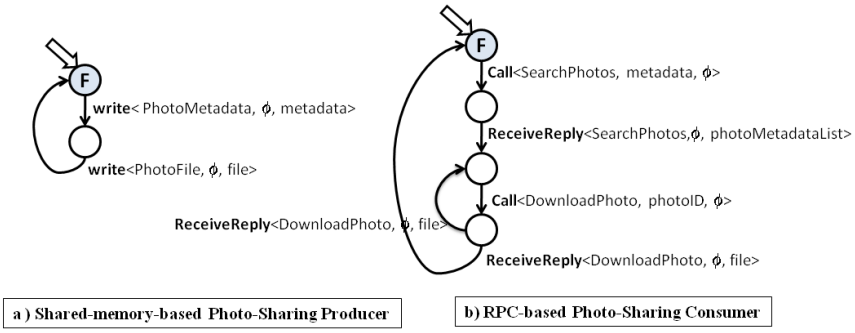

Fig. 3. Heterogeneous protocols in pervasive photo sharing

Let Act be the set of observable input/output actions and τ be the silent action (we use the usual convention that the output actions are denoted by an overbar while the input actions have no overbar). An extended LTS, which makes final states explicit, is a quintuple (S, L, D, F, s_0) where: (i) S is a finite set of states, (ii) $L \subseteq Act \bigcup \{\tau\}$ is a finite set of labels called the alphabet of the LTS, (iii) $D \subseteq S \times L \times S$ is a transition relation, (iv) $F \in S$ is the set of final states, and (v) $s_0 \in S$ is the initial state.

As an illustration, Figure 3 depicts the LTSs of the affordance protocols associated with the Photo-Sharing scenario that we informally introduced in Section 2.1.

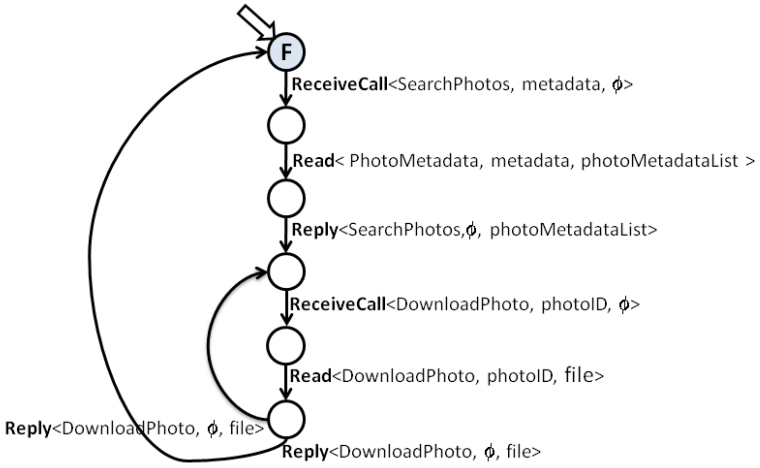

Fig. 4. Mediator protocol in pervasive photo sharing

In particular, an action of Act is specifically structured as: $MW< AP$, IN, $OUT >$, where MW denotes the *middleware function* that is called to interact with the peer system through the *application function* AP that is parameterized by *input* and *output parameters*, IN and OUT respectively.

Given the two LTSs, $\mathcal{PR}_1$ and $\mathcal{PR}_2$, characterizing the behaviors of functionally matching affordances, they are translated into LTSs $\mathcal{PR}'_1$ and $\mathcal{PR}'_2$ for the sake of comparison. The translated protocols are defined in a middleware-agnostic way over common application actions following application ontology alignment.

The translation of protocols in particular relies on the alignment of application layer functions into common application-specific ontologies (O in Figure 2) and on the translation of middleware functions from reference middleware ontology ($\mathcal{O}_{\mathcal{MW}}$ in Figure 2) into primitive send/receive actions [1].

Once $\mathcal{PR}'_1$ and $\mathcal{PR}'_2$ have been produced, we check their *compatibility* (also referred to as *mediated matching*) according to the set of traces T_1 and T_2 as-

sociated with $\mathcal{PR}'_1$ and $\mathcal{PR}'_2$, respectively. If the two protocols are compatible, then we are able to synthesize a mediator $\mathcal{M}$ that is such that when building the parallel composition $\mathcal{PR}_1||\mathcal{PR}_2||\mathcal{M}$, $\mathcal{PR}_1$ and $\mathcal{PR}_2$ are able to coordinate by reaching their final states.

Figure 4 shows the mediator protocols synthesized by our approach for the Photo-Sharing example.

4 Automated Monitoring of Mediators

The synthesis procedure described in the previous section yields a mediator that is able to bridge behavioral mismatches between NSs. However, affordance descriptions includes policies, which are used to express non-functional *requirements imposed by* the NS, or declarations of non-functional *characteristics*, *guaranteed by* the NS. In the Photo-Sharing example, a client may require that the time to obtain a list of photos that match a query must be less than X time units. This is an example of latency property. Similarly, the client may declare that it will never invoke the search operation more than three times in a minute (i.e., it guarantees it will generate a bounded workload).

In order to ensure that the CONNECTed system as a whole satisfies the requirements imposed by each NS participating in the connection, we adopt a runtime checking approach, supported by a dedicated monitoring infrastructure. This infrastructure (shown in Figure 5) is structured according to a generic, flexible architecture that decouples business-level (or high-level) event specification from the underlying observation and detection mechanisms. From a technical perspective, this decoupling is achieved by delegating to a *probe*, paired with mediator, the task of collecting low-level (i.e., primitive) event occurrences, which happen when a transition on the mediator LTS is taken. Primitive event occurrences are collected from the probes through a message-oriented backbone. The detection of complex events, defined as combinations of primitive events, is done using a Complex Event Recognizer [15]. Finally, complex event occurrences are notified to the interested consumers, again using the message-oriented backbone.

The monitoring manager is responsible for converting non-functional constraints coming from affordance specifications into directives to derive the probe(s), to instruct the Complex Event Recognizer, and to configure the routing of monitoring information over the monitoring bus (i.e., from the probes to the event recognizer and then to the consumers interested in specific complex events).

As an example, in this paper we consider a constraint that defines the acceptable latency for operations used by the RPC photo-sharing client. As already mentioned, other properties can be checked using the same framework, as long as they can be translated onto the complex event specification language used in the CONNECT monitoring system.

We assume that latency constraints are expressed in terms of operations belonging to the interface of the NS. For example, the already mentioned constraint imposed by the RPC client on the time it takes to get a list of photos that match a query, can be expressed as:

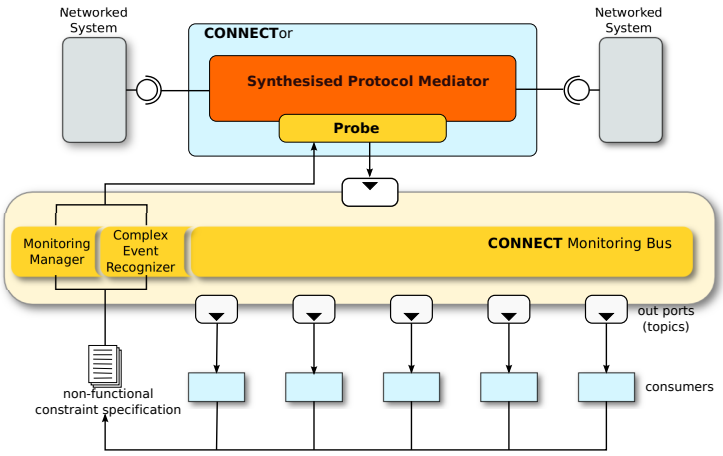

Fig. 5. CONNECT monitoring infrastructure

$$\Delta(Call(SearchPhotos,\text{-},\text{-}),ReceiveReply(SearchPhotos,\text{-},\text{-}) < X$$

which means "the time elapsed from calling the operation SearchPhotos to the receiving a reply for that invocation must be less than X time units".

This policy is expressed in terms of high-level functionalities (i.e., operations included in the public interface of the NS) and using the RPC middleware primitives (in this example, *Call, ReceiveReply*). It must be processed in order to derive a specification that is used to configure the monitoring system so that the policy can be checked.

In order to do so, the policy is converted into its corresponding formulation in terms of *complementary* actions performed by the mediator. In this example, *Call(SearchPhotos,-,-)* translates to *ReceiveCall(SearchPhotos,-,-)* and analogously *ReceiveReply(SearchPhotos,-,-)* translates to *Reply(SearchPhotos,-,-)*. The resulting constraint on the behavior of the mediator is therefore:

$$\Delta(ReceiveCall(SearchPhotos,\text{-},\text{-}),Reply(SearchPhotos,\text{-},\text{-}) < X'$$

Although in general, $X' = X + X_n$, where X_n is the delay due to the network, for the sake of simplicity, here we assume that this delay is negligible (i.e., $X_n = 0$) and therefore $X' = X$.

In real-life scenarios, X_n is typically not negligible, so a client policy that requires the delay to be less than X (observed on the client) translates into a requirement that the delay observed on the mediator be less than $X - X_n$. This is a general problem entailed by observing latency (or other time-related properties) in networked settings. However, this is beyond the scope this paper and we do not discuss it further.

```
1  rule "PhotoSearchLatency"
2  when
3    // this is the complementary of Call
4    $call:  ReceiveCall(
5      operation == "SearchPhotos";
6      $session : session_id)
7      from entry-point "PhotoSharingMediator"
8    $reply: Reply(
9      session_id == $session;
10     operation == "SearchPhotos";
11     this after[1200ms] $call )
12     from entry-point "PhotoSharingMediator"
13 then
14   // inject alarm on the monitoring bus
15 end
```

Listing 1.1. Sample rule for checking a latency policy

Finally, the expression obtained for the constraint on the mediator is translated into a language that is readily understood by the event-correlation engine. The example in Listing 1.1 shows the latency constraint expressed the specification language used by Drools Fusion [6], an open source rule engine with complex event processing capabilities.

The rule *PhotoSearchLatency* matches a *ReceiveCall* event followed by a *Reply event* (lines 4 and 8 respectively), ensuring that both refer to the same session (lines 6 and 9) and that the latter happens no earlier than 1200 ms [5] after the former (line 11). If all these conditions are verified, an alarm is injected into the monitoring bus (line 14) so that the subscribers for that kind of complex event can be notified.

5 Conclusion

The high degree of heterogeneity in the current digital system's underlying infrastructures thwarts the realization of the long-standing Ubiquitous Computing vision. Interoperability is a key requirement in such systems, where both functional and non-functional aspects expressed by Networked Systems have to be met while making them able to work together.

The continuous evolution characterizing the Ubiquitous environment, asks for on-the-fly approaches rather than relying on interoperability mechanisms fixed a-priori that are not adequate to completely address the problem.

In order to achieve a complete CONNECTion (functional and non-functional interoperability), this paper presented a combined interoperability approach.
It is made by the integration of an automated technique for the synthesis of

[5] In this example, we assume 1200 ms is the concrete value for X'

mediators with a monitoring mechanism. The mediators provide functional interoperability and the monitors make it possible to assess the non-functional characteristics of the connected system at runtime that cannot be assessed statically at synthesis time.

As future work, we plan to investigate the following aspects that are important in the larger CONNECT picture [2].
We need to propose a language to express non-functional constraints and properties.
We need to provide reaction policies or reaction policy patterns that can be undertaken when something wrong is detected by the monitoring. Examples are: to use predictive approaches that try to prevent the wrong behaviors; to adapt the CONNECT architectural infrastructure, if possible, for improving the provided connection; eventually, to notify the Networked Systems about the unexpected behavior, and let them directly handle the problem.
As a long-term goal, we will work towards including reasoning about non-functional properties into the synthesis process [8].

References

1. A. Bennaceur, G. Blair, N. Georgantas, P. Grace, P. Inverardi, V. Issarny, A. Pathak, R. Saadi, and R. Spalazzese. Revisiting the Middleware Paradigm: On-the-fly Interoperability in Highly Complex Distributed Systems. *Technical Report, INRIA Rocquencourt - Paris*, May 2010.
2. A. Bennaceur, G. S. Blair, F. Chauvel, N. Georgantas, P. Grace, F. Howar, P. Inverardi, V. Issarny, M. Paolucci, A. Pathak, R. Spalazzese, B. Steffen, and B. Souville. Towards an architecture for runtime interoperability. In *Proceedings of ISoLA 2010 - 4th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*. Springer, 2010.
3. A. Bertolino, F. Di Giandomenico, A. Di Marco, V. Issarny, F. Martinelli, P. M. Masci, I. Matteucci, R. Saadi, and A. Sabetta. Dependability in dynamic, evolving and heterogeneous systems: the CONNECT approach. In *2nd International Workshop on Software Engineering for Resilient Systems SERENE 2010*, London, U.K., 2010.
4. A. Bertolino, P. Inverardi, P. Pelliccione, and M. Tivoli. Automatic synthesis of behavior protocols for composable web-services. In *ESEC/FSE '09: Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 141–150, New York, NY, USA, 2009. ACM.
5. D. Bianculli and C. Ghezzi. Monitoring conversational web services. In *IW-SOSWE '07: 2nd international workshop on Service oriented software engineering*, pages 15–21, New York, NY, USA, 2007. ACM.
6. P. Browne. *JBoss Drools Business Rules*. Packt Publishing, 2009.
7. G. Costa and I. Matteucci. Enforcing private policy via security-by-contract. *Special issue Identity and Privacy Management. UPGRADE Journal*, Vol. XI,(1):43–53, February 2010.
8. F. Di Giandomenico, M. Kwiatkowska, M. Martinucci, P. Masci, and H. Qu. Dependability analysis and verification for connected systems. In *Proceedings of*

ISoLA 2010 - 4th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation. Springer, 2010.
9. J. J. Gibson. *The ecological approach to visual perception*. Houghton Mifflin, 1979.
10. F. Howar, B. Jonsson, M. Merten, B. Steffen, and S. Cassel. On handling data in automata learning: Considerations from the connect perspective. In *Proceedings of ISoLA 2010 - 4th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*. Springer, 2010.
11. P. Inverardi, V. Issarny, and R. Spalazzese. A theory of mediators for eternal connectors. In *Proceedings of ISoLA 2010 - 4th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*. Springer, 2010.
12. P. Inverardi and L. Mostarda. Desert: a decentralized monitoring tool generator. In *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 529–530, New York, NY, USA, 2007. ACM.
13. V. Issarny, B. Steffen, B. Jonsson, G. Blair, P. Grace, M. Kwiatkowska, R. Calinescu, P. Inverardi, M. Tivoli, A. Bertolino, and A. Sabetta. CONNECT Challenges: Towards Emergent Connectors for Eternal Networked Systems. In *14th IEEE International Conference on Engineering of Complex Computer Systems*, Postdam Germany, 2009.
14. R. M. Keller. Formal verification of parallel programs. *Commun. ACM*, 19(7):371–384, 1976.
15. D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., 2001.
16. R. Spalazzese, P. Inverardi, and V. Issarny. Towards a formalization of mediating connectors for on the fly interoperability. In *Proceedings of the Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture (WICSA/ECSA 2009)*, pages 345–348, 2009.
17. R. Vaculín and K. Sycara. Towards automatic mediation of OWL-S process models. *Web Services, IEEE International Conference on*, 0:1032–1039, 2007.
18. M. Weiser. The computer for the 21^{st} century. *Scientific American*, Sep. 1991.
19. S. K. Williams, S. A. Battle, and J. E. Cuadrado. Protocol mediation for adaptation in semantic web services. In *ESWC*, pages 635–649, 2006.